基于 EBNF 和二次爬取策略的 XSS 漏洞检测技术 *

黄文锋 [1]，李晓伟 [2]，霍占强 [2†]

(1. 河南省科学技术信息研究院，郑州 450003; 2. 河南理工大学 计算机科学与技术学院，河南 焦作 454000)

摘　要：跨站脚本（XSS）攻击是目前互联网安全的最大威胁之一。针对传统基于渗透测试技术的漏洞检测方法中攻击向量复杂度低易被过滤、整体检测流程繁琐等问题，提出了一种基于扩展的巴科斯范式（EBNF）的攻击向量自动生成方法和 XSS 漏洞二次爬取策略。通过定义 EBNF 规则生成规则解析树，按层次遍历获得高复杂度攻击向量。在首次爬取页面时，将输入点信息嵌入到攻击向量后请求注入，之后进行二次爬取，请求合法参数获得返回页面。最后设计实现了原型系统，并使用两个平台进行漏洞检测。通过对比实验证明，该系统检测流程简单，在一定程度上提高了漏洞检测数，降低了漏洞误报率。

关键词：跨站脚本；扩展的巴科斯范式；攻击向量；渗透测试

中图分类号：TP309　　doi: 10.3969/j.issn.1001-3695.2018.02.0170

XSS vulnerability detection technology based on EBNF and twice crawling strategy

Huang Wenfeng[1], Li Xiaowei[2], Huo Zhanqiang[2†]

(1. *Henan Provincial Institute of Scientific & Technical Information*, *Zhengzhou* 450003, *China*; 2. *College of Computer Science & Technology Henan Polytechnic University*, *Jiaozuo Henan* 454000, *China*)

Abstract: Cross-site scripting (XSS) attacks have been one of the biggest threats to Internet security. Aiming at the problems of traditional vulnerability detection method based on penetration testing technology, such as attack vectors of low complexity easy to filter and overall detection process cumbersome, this paper proposed a new attack vectors automatic generation method which based on extended Backus-naur form (EBNF) and a XSS vulnerability twice crawling strategy. By defining the EBNF rule, the method generated a rule-parsing tree, and then traversed hierarchically the tree to obtain high-complexity attack vectors. In the first page crawling, the strategy inserted input point information to attack vectors and requested injection. Then it carried on the second crawling and requested legal parameters to get the return page. In the final, this paper designed and implemented a prototype system, and used two platforms for vulnerability detection. The comparative experiments prove that the system has a simple detection process, and to a certain extent, improves the number of vulnerability detection and reduces the false positive rate.

Key words: XSS; EBNF; attack vector; penetration testing

0　引言

从互联网诞生以来，网络安全问题就始终存在。随着 Web 应用程序的推广，网络安全问题也越来越被安全研究者所重视。开放 Web 应用安全项目（OWASP）发布的 2017 年十大最关键的 Web 应用安全风险的最新版本[1]中跨站脚本（XSS）稳居其中，排在第七位。WhiteHat Security 组织在最新发布的 2017 年应用程序安全统计报告[2]中对所有安全风险进行了统计，结果显示，XSS 在所有的安全风险中比重高达 33%，仅次于占比 37% 的信息泄露风险。XSS 作为第二普遍的安全问题，近三分之二的网络应用存在着 XSS 风险，一旦遭受攻击就会造成极大的损害。由此可知，XSS 安全风险对互联网安全产生的威胁已不容忽视。

目前，国内外研究者针对 XSS 的漏洞检测的工具大致有黑盒测试工具[3]和白盒测试工具[4]两种类型。黑盒测试工具主要使用动态分析技术[5]实现，白盒测试工具则主要使用静态分析技术[6]实现，也有部分研究者使用动静分析结合的技术[7]实现漏洞的检测。其中，动态分析技术是在网络应用程序运行过程中进行检测的技术；常用的分析方法包括渗透分析、动态污点分析、流量分析以及监控、过滤等。静态分析技术是通过审查

收稿日期：2018-02-04；**修回日期：**2018-04-15　**基金项目：**国家自然科学基金资助项目（61472342，61572379）；河南省高等学校重点科研计划项目（17A520007）

作者简介：黄文锋（1966-），男，河南郑州人，高级工程师，主要研究方向为网络安全；李晓伟（1992-），男，河北邯郸人，硕士研究生，主要研究方向为网络与信息安全；霍占强（1979-），男（通信作者），河北邯郸人，副教授，博士，主要研究方向为网络性能分析（hzq@hpu.edu.cn）.

网络应用程序的源代码或字节代码来检测漏洞；常用的分析方法包括静态污点分析、符号执行和字符串分析等。为了避免网站开发过程中产生 XSS 漏洞，有研究者提出了安全编程和模型化等解决方案。

李威等人[8]提出了一种专门针对存储型 XSS 漏洞的检测方法，使用 BNF 范式自动生成初始攻击向量，并再次进行变异操作；之后使用聚焦网络爬虫爬取页面，对爬取到的 form 表单通过提交探子向量并进行全网扫描寻找输出点；最后根据匹配到的输入输出点注入攻击向量并进行漏洞检测。该方法虽然减少了漏洞检测过程中的测试次数，但是使得检测之前的准备流程变得过于繁琐，在自动生成初始攻击向量后仍需要执行变异操作，并且使用探子向量寻找每一个 form 表单输出点时都需要进行全网扫描。顾明昌等人[9]对渗透测试方法做了一定程度的改进，提出了一种基于符号集的 XSS 攻击向量自动生成方法，并使用决策树分类算法对攻击向量进行分类，以便在检测阶段进行测试输入点时直接按类别使用。该方法在渗透测试前同样通过使用探子向量去除一部分不存在漏洞的 URL，以减少不必要的测试，提高检测效率。但是探子向量会导致爬取过程变得复杂，而且该方法生成的攻击向量比较简单，很容易被服务器端过滤。吴子敬等人[10]提出了一种反过滤规则集对 XSS 攻击向量进行转换，从而绕过服务器端对不同类型恶意代码的过滤机制，并通过自动爬虫实现 XSS 攻击向量的自动注入，这也是对渗透测试方法的一种改进。然而该方法只对如何能够使攻击向量成功绕过服务器端的过滤机制进行了研究，而在漏洞检测过程中，仅仅通过爬虫注入一次攻击向量后就开始检测漏洞，容易造成大量存储型漏洞的漏报。

本文提出一种新的攻击向量自动生成方法，在渗透测试技术的基础上，设计出一款漏洞检测系统。该系统通过使用 EBNF 范式定义攻击向量生成规则，增加 XSS 攻击向量生成的随机性，提高攻击向量的复杂度以便绕过服务器，并在恶意脚本片段中嵌入输入点信息来记录输入点，最终直接得到变异的攻击向量。作为一种黑盒测试工具，该系统在待检测系统运行期间进行了两次页面爬取操作。在首次爬取页面过程中注入精心设计的攻击向量，随后在首次爬取结果的基础上有策略进行二次爬取，最后对两次爬取保存下来的页面使用正则匹配方法进行分析检测，根据恶意脚本片段的输入点信息直接定位漏洞。

在设计和实现本文提出的 XSS 漏洞检测系统时，没有考虑 WAF 防火墙和反爬虫措施对检测结果的影响，其原因主要有两点：a)本系统面向的使用群体是网站开发者和维护人员，目的是尽可能多地检测出自己设计开发的待检测 Web 系统的 XSS 漏洞，从而进行修复，如果 Web 系统存在 WAF 防火墙或者反爬虫的措施，则需要暂时禁用；b)本文不是专业的爬虫系统，侧重点在于设计出高复杂度的攻击向量，从而绕过服务器端的过滤机制。爬取页面方面，通过改进现有开源爬虫工具，使用二次爬取策略，以降低存储型漏洞的漏报率。

1　相关知识

1.1　XSS 漏洞

到目前为止，研究者将 XSS 漏洞大致分为三种[11~13]，即反射型 XSS（reflected-XSS）漏洞、存储型 XSS（stored-XSS）漏洞、基于 DOM 型 XSS（DOM based XSS）漏洞。

反射型 XSS 和存储型 XSS 漏洞是服务器端漏洞，其形成原理是：攻击者在浏览器端提交恶意代码到服务器端，服务器端未能进行有效的过滤和验证，导致恶意代码出现在返回的页面中，此时用户访问该页面时就会加载恶意代码，产生漏洞。基于 DOM 型 XSS 是反射型 XSS 和存储型 XSS 的变体，与存储型 XSS 相比有着很大的不同，但与传统的反射型 XSS 相比差别很微妙，这微妙的差别却导致了攻击特性的彻底不同。基于 DOM 型 XSS 中服务器没有直接将恶意脚本作为页面的一部分返回，而是在浏览器加载页面时，由于页面中合法脚本直接将用户的输入数据作为 HTML 内容输出到页面，所以在合法脚本执行后，恶意脚本也随之被插入到页面之中。

1.2　渗透测试

渗透测试技术是一种尽可能完整地模拟黑客使用的漏洞发现技术和攻击手段，对目标网络的安全性作深入的测试进而发现网络存在的任何弱点、技术缺陷或漏洞的技术。渗透测试方法类型很多，但当前业界最通用并被广泛接受的两种渗透测试方法是黑盒测试和白盒测试。

黑盒测试：又称为外部测试，测试人员完全处于对系统一无所知的状态，通过使用信息采集工具从 DNS、Web、Email 及各种公开对外的服务器上获取信息，模拟真实的黑客技术，有组织有步骤地对目标网络进行逐步的渗透与入侵，发现目标网络中一些已知或未知的安全漏洞，并进行安全性评估。

白盒测试：又称为内部测试，测试人员事先获取到关于目标网络的所有内部和底层的信息，并以最小的代价查看和评估目标网络中最严重的安全漏洞。白盒测试能够消除几乎所有存在于目标网络内部代码和设施中的安全隐患，从而使其能够更加牢固地抵挡来自外部的恶意攻击。

1.3　EBNF 范式

EBNF（extended Backus-naur form）是一种元语法符号表示法，主要用来正式定义计算机编程语言的语法以及其他许多语言的语法，是基本巴科斯范式 BNF（Backus-naur form）的扩展。

EBNF 定义了把各符号序列分别指派到非终结符的产生规则，如 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' 。

这个产生规则定义了在这个指派的左端的非终结符 letter。竖杠表示可供选择，而终结符被引号包围，最后跟着分号作为终止字符。所以 letter 是一个 a 或 b 直到 g 的一个英文字母。

2　基于 EBNF 的攻击向量设计

本章内容首先介绍如何设计攻击向量种子，之后介绍如何在攻击向量种子的基础上定义 EBNF 范式攻击向量生成规则，

最后根据规则生成变异的攻击向量。

2.1　攻击向量设计

在 XSS 漏洞检测过程中，设计的攻击向量其有效性是决定能否更加全面地检测出 XSS 漏洞的关键。攻击向量有很多种类型，并符合一定的组成方式，即满足以下几个规则：a)包含可执行的 JavaScript 脚本；b)符合 HTML 语法并可以嵌入到 HTML 页面中；c)能够被触发执行。通常一种类型的攻击向量不足以检测出所有可能存在的漏洞，因此需要使用多种类型重复检测。本文主要使用如下几种类型的攻击向量用来检测 XSS 漏洞：

a)直接使用脚本作为攻击向量。

<script>alert('xss')</script>

<script src="xss.js"></script>

……

b)把脚本作为 HTML 元素的属性值嵌入到 HTML 元素中作为攻击向量。

……

c)把脚本作为 HTML 元素的触发事件嵌入到 HTML 元素中作为攻击向量。

<body onload="alert('xss')">

……

d)把脚本作为 CSS 样式属性的 url 值嵌入到 HTML 元素中作为攻击向量。

<body style="background-image: url('javascript:alert("xss")');">

<style>@import url("javascript:alert('xss')");</style>

……

如果输入点是 HTML 标签内属性的值，则需要先将标签闭合，即在攻击向量中添加起始字符串“=">”或者“">”。

由于攻击向量 JavaScript 代码部分可以填写任意合法的 JavaScript 语句，所以本文在代码部分增加了输入点信息，这样就避免了在检测过程中记录输入点和输出点信息，从而很大程度上简化了漏洞检测的整个流程。比如说对于攻击向量<script>alert('xss')</script>，本文将 alert('xss')部分内容替换为了 alert(URL)，其中 URL 实际内容是标识输入点信息的 URL 字符串。

2.2　EBNF 规则定义

由于原有 EBNF 语法无法满足需求，所以本文在原有 EBNF 的基础上添加了几条新的语法规则。定义如下：

定义 1　定义新语法“<…>”，其内的每组元素必须以“|”连接，含有分组的概念，可以多层嵌套，在一个规则中如果出现由“,”连接两次或以上同一组“<…>”时，只能选取其中一对完全相同的分组，只出现一次的话则与“(…)”语法规则相同。

定义 2　定义新语法“:”，以符号“:”连接的元素排列是有序的，含有“|”的概念，一个规则中如果出现由“,”连接两次或以上此类连接字符串时，取值时按序对应取值，只出现一次的话与“|”语法规则相同。

在注入攻击向量时，复杂度不高的攻击向量通常会被服务器端的过滤机制净化掉，因此需要进行变异处理，增加攻击向量的复杂度，使之能够绕过服务器端的过滤机制，提高攻击向量的有效性。以攻击向量“<script>alert(URL)</script>”为例说明，常见的变异处理方式如表 1 所示。

表 1　变异处理方式

变异处理	变异结果
十进制 ASCII 编码	<script>alert(URL)</script>
十六进制 ASCII 编码	<script>alert(URL)</script>
HTML 编码	<script>alert(URL)</script>
URL 编码	%3cscript%3ealert(URL)%3c%2fscript%3e
JavaScript 编码	<script>\u0061\u006c\u0065\u0072\u0074(URL)</script>
混合编码	<script%3eale\u0072\u0074(URL)</script>
大写转换	<SCRIPT>alert(URL)</SCRIPT>
小写转换	<script>alert(URL)</script>
大小写混合转换	<scriPT>alert(URL)</SCriPt>

根据以上变异处理方法，本文定义了不同形式的 EBNF 规则，并将其分为基本字符规则和基本语句规则。基本字符规则如表 2 所示。

表 2　基本字符规则

<table>
<tr><th>基本字符</th><th>基本字符规则</th></tr>
<tr><td>26 个英文字母</td><td>a = 'a'|'A'|('&#','97'|'65'|('x'|'X','61'|'41'),';')|'%61'|'%41';
z = 'z'|'Z'|('&#','122'|'90'|('x'|'X','7a'|'7A'|'6a'|'6A'),';')|'%7a'|'%7A'|'%5a'|'%5A';
……</td></tr>
<tr><td>特殊符号</td><td>apostrophe_symbol = """|('&#','39'|('x'|'X','27'),';')|'%27';
quotation_mark_symbol = '"'|'"'|('&#','34'|('x'|'X','22'),';')|'%22';
……</td></tr>
</table>

基本语句建立在基本字符规则的基础上，其规则如表 3 所示。

表 3　基本语句规则

<table>
<tr><th>基本语句</th><th>基本语句规则</th></tr>
<tr><td>HTML 类型</td><td>html_element_two = <(a,c,r,o,n,y,m)|b|(b,i,g)|(c,i,t,e)|(c,o,d,e)|(d,f,n)|(e,m)|(h,'1'|'2'|'3'|'4'|'5'|'6')|i|(k,b,d)|(s,a,m,p)|(s,m,a,l,l)|(t,t)|(s,t,r,o,n,g)|(v,a,r)>;</td></tr>
</table>

<table>
<tr><td>其他类型</td><td>event_attr_keyboard = (o,n,k,e,y,d,o,w,n)|(o,n,k,e,y,p,r,e,s,s)|(o,n,k,e,y,u,p);
……
left_script_label = less_than_sign_symbol,script,greater_than_sign_symbol;
right_script_label = less_than_sign_symbol,solidus_symbol,script,greater_than_sign_symbol;
……</td></tr>
</table>

2.3 攻击向量变异

在基本字符规则和基本语句规则的基础之上，可以定义各种类型攻击向量的生成规则。根据上文介绍过的攻击向量类型定义如下几条规则，如表 4 所示。

表 4　攻击向量生成规则

<table>
<tr><th>生成方式</th><th>攻击向量生成规则</th></tr>
<tr><td>直接使用脚本</td><td>payload_rule0 = left_script_label,js_strings,right_script_label;
payload_rule6 = less_than_sign_symbol,script,src_value,(solidus_symbol,greater_than_sign_symbol)|(greater_than_sign_symbol,right_script_label)</td></tr>
<tr><td>脚本作为 HTML 元素的属性</td><td>payload_rule3 = left_label_url,url_attr,equals_sign_symbol,url_value,(solidus_symbol,greater_than_sign_symbol)|(greater_than_sign_symbol,right_label_url);</td></tr>
<tr><td>脚本作为 HTML 元素的事件</td><td>payload_rule5 = left_label_event,event_attr,equals_sign_symbol,quotations,js_strings,quotations,greater_than_sign_symbol,[right_label_event];</td></tr>
<tr><td>脚本作为 CSS 样式属性</td><td>payload_rule1 = left_style_label,at_sign,import,url,left_parenthesis_symbol,url_value,right_parenthesis_symbol,right_style_label;
payload_rule2 = left_style_label,html_element_style,left_curly_brace,css_url_attr,colon_symbol,url,left_parenthesis_symbol,url_value,right_parenthesis_symbol,right_curly_brace,right_style_label;
payload_rule4 = left_label_style,style_attr,equals_sign_symbol,css_url_attr,colon_symbol,url,left_parenthesis_symbol,url_value,right_parenthesis_symbol,greater_than_sign_symbol,[right_label_style];</td></tr>
<tr><td>闭合 HTML 标签</td><td>payload_rule7 = [[equals_sign_symbol],quotations,greater_than_sign_symbol],payload_rule0|payload_rule1|payload_rule2|payload_rule3|payload_rule4|payload_rule5|payload_rule6;</td></tr>
</table>

按照这些规则进行解析之后，可以生成复杂度很高的变异攻击向量。这里使用规则解析树来对其进行解析，每条规则对应生成一棵规则解析树，之后按照一定的规则对解析树进行层次遍历便可以生成变异的攻击向量。例如基本字符规则“a = 'a'|'A'|('&#','97'|'65'|('x'|'X','61'|'41'),';')|'%61'|'%41';”对应的规则解析树如图 1 所示。

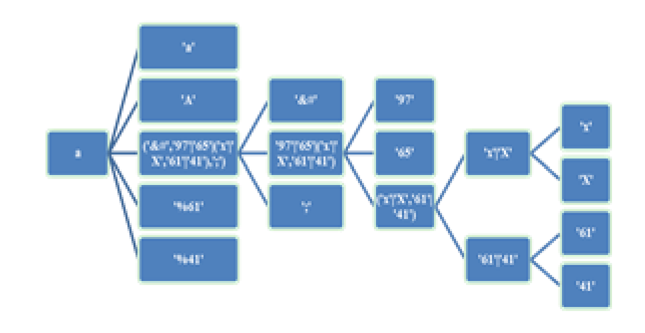


图 1　基本字符规则解析树举例

同理，以基本语句规则“payload_rule0 = left_script_label, js_strings,right_script_label;”为例，其对应的规则解析树如图 2 所示。该解析树的每一个叶子节点都是一棵由基本字符规则生成的解析树的根节点。

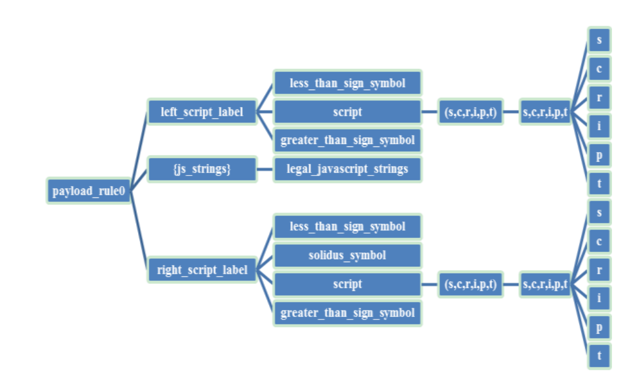


图 2　基本语句规则解析树举例

根据每种攻击向量的生成规则，对应的解析树生成的变异攻击向量如表 5 所示。

表 5　变异后攻击向量

变异规则	变异后攻击向量
payload_rule0	<S%63r%49pt>alert(URL)%3C%2fSC%72%69p%54%3E
payload_rule1	<STy%6c%45%3e@%69m%50%4F%52%54+ +%55RL("%6AA%56%61s%63%72ipT+%3aalert(URL)")<%2fs%54%79%4Ce>
payload_rule2	<%73Ty%6CE>%41%7b%6c%69St%2DS%54%59lE-%49m%41ge%3A%20ur%6C(%22%6AAV%61%53cRI%50T%20%3Aalert(URL)")%7D<%2fS%74yl%65>
payload_rule3	<%69m%67+ LO%4Eg%64%45s%63%3D%22j%41%76a%53%63%52%69Pt:alert(URL)%22></%49%4dg>
payload_rule4	<a S%54y%6C%65=bA%43kgRo%75ND%2dI%4D%41%47%6

	5%3AUrL("%4aA%76A%53%43Ript%20%20:alert(URL)")%3E
payload_rule5	%3c%61%43r%4f%4e%59%6d +%4fN%4BEY%75p="alert(URL)">%3c/a%63%72%4F%6ey%6d%3E
payload_rule6	%3c%53%43R%49%70t%20sR%43%20%3d"http://xss.ha.ckers.org/a.js"%3E%3c/scR%49pT>
payload_rule7	%3D"><sc%72%69pT>alert(URL)%3C/sC%72%49Pt>

3 检测系统框架与流程

3.1 整体框架

根据前文对各种类型 XSS 漏洞原理的研究分析，本文设计出一款针对存储型和反射型 XSS 漏洞的检测系统，目的是生成高复杂度攻击向量，并实现简化 XSS 漏洞检测流程的功能。本系统一共有三大模块组成，整体结构如图 3 所示。其中攻击向量生成模块包括规则解析树生成子模块和变异攻击向量生成子模块两个子模块，爬虫模块包括页面爬取子模块和页面解析子模块两个子模块。攻击向量生成模块是检测系统的基础模块，为爬虫模块和漏洞检测模块提供服务，主要功能是生成变异后的攻击向量。在该模块中，规则解析树生成子模块根据定义的 EBNF 规则生成与其相对应的规则解析树，之后变异攻击向量生成子模块根据规则解析树生成不同规则下的变异后攻击向量字符串。爬虫模块是检测系统的核心模块，为漏洞检测模块提供服务，主要功能是实现待检测系统所有页面的爬取和解析，两次页面爬取过程都在此模块进行。其中，页面爬取子模块用来生产爬虫并爬取页面，页面解析子模块用来解析爬虫爬取到的页面并将分析结果反馈给爬虫。漏洞检测模块是检测系统的最终模块，依赖前置模块提供的服务，通过对保存的页面内容进行检测从而判定漏洞。

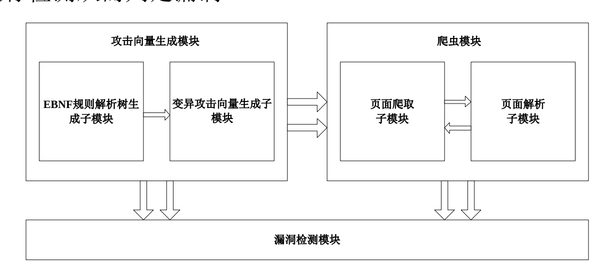


图 3 系统整体结构

3.2 系统流程

本系统检测方法是基于渗透测试技术的 XSS 漏洞检测方法。整个检测流程如图 4 所示。

整个漏洞检测流程分为了五个子流程，分别为攻击向量生成子流程、初始化爬虫子流程、首次爬取与解析子流程、二次爬取与解析子流程以及漏洞检测子流程。首先使用上文所述的基于 EBNF 的攻击向量生成方法生成不同类型的变异攻击向量，便于后续流程的操作使用；之后初始化爬虫相关配置，精心挑选种子 URL 加入待爬取 URL 队列中；爬虫初始化完成以后，使用漏洞爬虫进行首次的页面爬取与解析过程，保存一部分已爬取页面；然后进行二次页面爬取与解析流程，获得首次未爬取到的页面；最后对所有爬取到的页面进行漏洞检测，判定是否存在 XSS 漏洞。下面对每个子流程的具体执行过程一一作详细介绍。

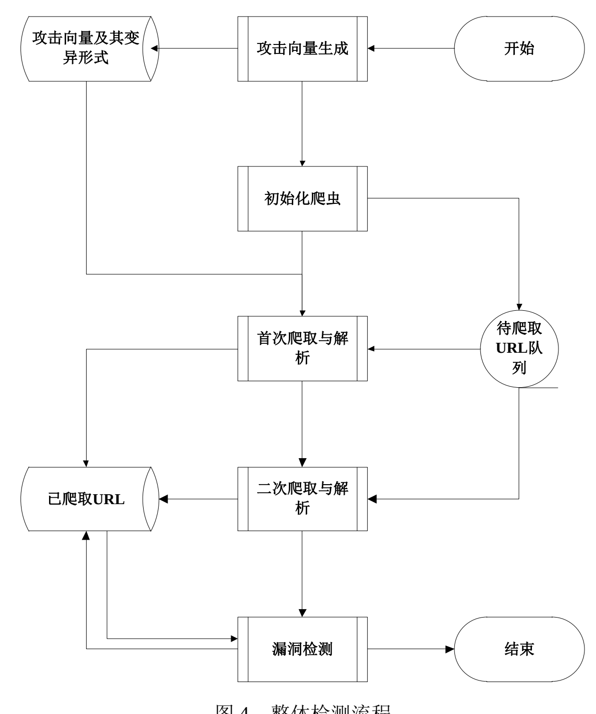


图 4 整体检测流程

3.2.1 攻击向量生成

该子流程实现了变异攻击向量自动生成的功能，得到的攻击向量复杂度高，能够有效绕过服务器端的过滤机制。具体执行流程如图 5 所示。首先初始化攻击向量生成的配置信息；然后依照配置信息，解析 EBNF 格式的攻击向量生成规则文件，生成变异的攻击向量。

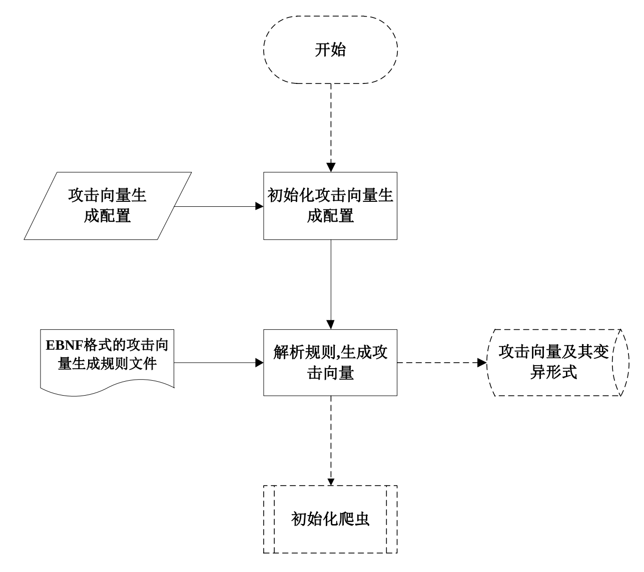


图 5 攻击向量生成子流程

3.2.2 初始化爬虫

在此子流程中，为了使爬虫能够爬取到系统更深层次的页面，首先需要在待爬取系统中进行登录认证；认证过后，对爬虫作一些基本配置，如爬虫数目和爬取深度等；最后将精心挑选的一部分种子 URL 加入待爬取 URL 队列中。详细流程如图 6 所示。

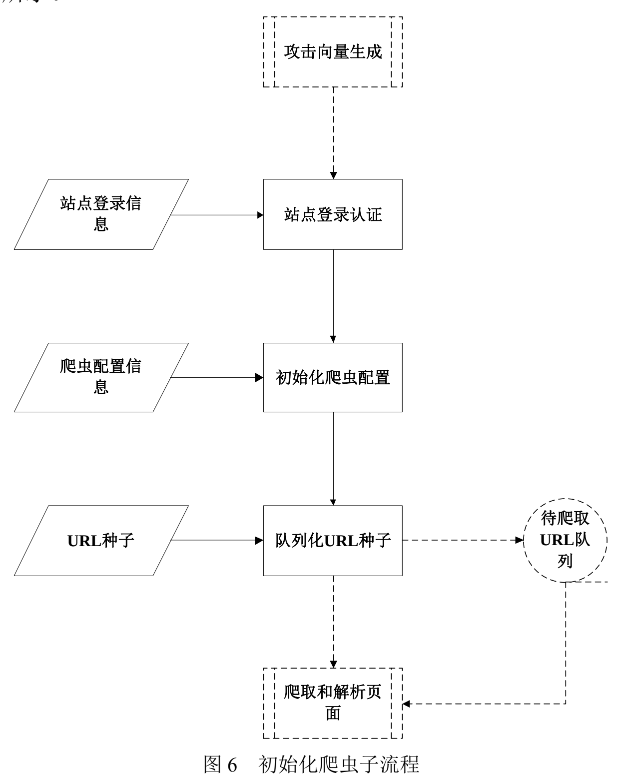


图 6　初始化爬虫子流程

3.2.3 爬取与解析

爬虫爬取待检测系统页面漏洞时，会存在这样一种情况：系统存在一部分具有删除功能的 URL 请求，这些 URL 请求会删除一些服务器端数据库存储的数据。由于页面爬取顺序的原因，爬虫首先通过某个存储型漏洞的输入点将攻击向量注入到服务器端数据库中，之后又在爬取到对应的输出点页面前，爬取了具有删除此条信息功能的 URL 请求，这样就导致数据库之前注入的攻击向量信息被删除，从而影响到后续检测页面漏洞的流程，造成存储型漏洞的漏报。

因此，本系统在解析页面时通过使用常规解析结合语义分析的方式，在抽取新的 URL 请求时进行语义判断，将包含有“del”“delete”“remove” 和 “rmv” 等具有删除移除语义的 URL 进行过滤。同时，提取出 URL 对应的标签词，并将包含有 “删除”“delete”“移除”“remove”和“删掉”等词汇的 URL 过滤。例如，某个 HTML 页面中含有这样一个超链接：删除，此条 URL 对应的标签词为“删除”，URL 中又包含有“del”字符串，因而将该 URL 过滤不进行后续爬取。由于具有删除功能的 URL 请求不会存在反射型或存储型 XSS 漏洞，所以是否爬取不会影响到后续 XSS 漏洞的检测。

本系统一共有两次爬取与解析过程，其中二次爬取相比较首次爬取流程简化许多，是在首次爬取结果的基础上进行爬取。两次爬取主要针对的漏洞类型不完全相同，首次爬取主要偏向于反射型漏洞的爬取，存储型漏洞可能只爬取到小部分；二次爬取则是针对于存储型漏洞，不再进行反射型漏洞的爬取。首次爬取流程如图 7 所示。

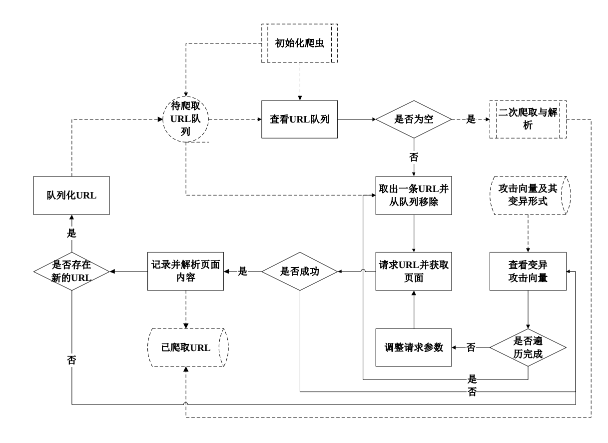


图 7　首次爬取与解析子流程

在首次爬取过程中，从待爬取 URL 队列中取出一条 URL 之后，首先使用合法参数发送一次请求，获取返回页面并解析，目的是抽取出新的待爬取 URL 加入队列；之后调整请求参数，注入变异后的攻击向量；重复该次请求，并保存返回页面。循环这一过程，直到所有类型的攻击向量注入完成。为了能够成功请求获取返回页面，系统在解析页面时对 URL 请求参数进行了语义分析，同样使用请求参数对应的标签词以及参数的 key 字符串进行判断，以保证请求参数的合法性。例如，某个请求参数对应的标签词为“预约时间”，可以判断该参数的合法输入应该是日期格式，或者说某个请求参数的 key 字符串为 “startSalesAmount”，可以判断该参数的合法输入则是数字格式等。

对于反射型漏洞，在请求后获取到的页面中就包含有攻击向量内容，只需要进行一次爬取即可。然而对于存储型漏洞来说，由于爬取时待爬取 URL 的优先级和待检测系统相关性很高，所以定义待爬取 URL 的优先级十分困难，这就导致爬取顺序具有很大的随机性，无法保证上次请求在注入攻击向量后，下次请求就能够获取到对应的输出点页面。因此本文增加了二次爬取过程来解决这一问题。详细执行流程如图 8 所示。

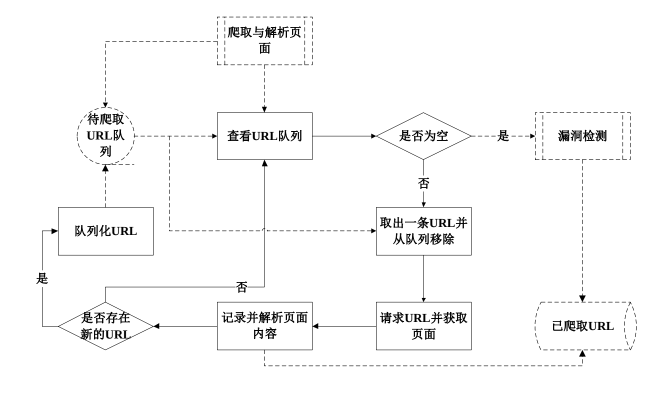


图 8　二次爬取与解析子流程

二次爬取时，不再对请求参数作调整，直接使用系统爬取

到的 URL 参数进行请求。由于在首次爬取过程中已通过存储型漏洞将攻击向量注入到服务器端，所以二次爬取到的页面必定包含有输出点页面。又由于首次爬取注入的攻击向量中包含有输入点信息，这样就可以保证输入点和输出点的一致性。

3.2.4 漏洞检测

在进行两次爬取与解析过程之后，对待检测系统的每一次请求，其获取到的页面都进行了保存。漏洞检测过程只需要遍历这些页面，分析页面内容中是否包含有被注入的攻击向量信息。如果存在，则认定为 XSS 漏洞，然后根据攻击向量中含有的输出点信息记录漏洞信息；若不存在则忽略。具体流程如图 9 所示。

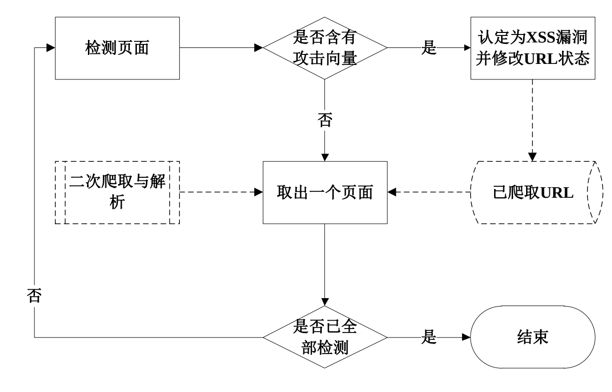


图 9　漏洞检测子流程

4　实验分析

为了验证本系统在检测 XSS 漏洞方面的可行性和有效性，本文设计了两组对比实验进行验证。实验指标使用漏洞检测数、漏洞漏报率和漏洞误报率。检测数表示实际检测到的漏洞数目。检测数的计算只与 URL 相关，与请求参数不相关，即一条 URL 即使存在多个能够注入攻击向量的请求参数，检测数也只计算一次。漏洞漏报率是检测系统未发现的漏洞和实际漏洞数的比值；误报率则反映了检测到的漏洞的准确性。

第一组对比实验使用本文检测系统和 AWVS（acunetix Web vulnerability scanner）漏洞扫描工具对银行间业务信息共享与交流平台进行检测。其中 AWVS 工具是一个商业化漏洞扫描软件，包含有收费版和免费版两种版本。本文使用免费版进行对比实验。信息共享与交流平台是一个标准 JSP+Struts2+MySQL+Hibernate 架构的 B/S 系统，目前已投入使用。由于该系统漏洞情况是未知的，所以使用漏洞检测数和漏洞误报率作为实验指标。实验数据对比结果如表 6 所示。

表 6　实验 1 对比结果

实验指标	本文系统	AWVS 工具
漏洞检测数	36	2
漏洞误报率	0	0

第二组对比实验使用本文检测系统和 AWVS 工具对 XSS 漏洞实验平台进行漏洞检测。XSS 漏洞实验平台是自行搭建的用于存储型和反射型漏洞检测的平台，其漏洞情况已知，实验指标使用漏洞检测数、漏洞漏报率和漏洞误报率。对比实验结果如表 7 所示。

表 7　实验 2 对比结果

实验指标	本文系统	AWVS 工具
漏洞检测数	14	9
实际漏洞数	16	16
漏洞漏报率	12.50%	31.25%
漏洞误报率	0	6.25%

从实验 1 的结果可以看出，在未知漏洞的情况下，本系统的检测数要高很多；实验 2 的结果显示，本系统的漏报率要低于 AWVS 工具，造成以上结果的原因是 ENBF 范式生成的攻击向量复杂度非常高，可以有效绕过服务器端的验证和过滤机制，从而注入恶意代码。由于系统忽略了具有删除功能的 URL，所以对于存储型漏洞注入的攻击向量，在爬取过程中不会被删除，进而能够在二次挖掘过程中检测出大量存储型漏洞，漏洞漏报率得到很大程度的降低。由两组对比实验结果可以发现，本系统两次测试的误报率都是 0，这是由于攻击向量中包含有输入点信息，这样就确保了进行漏洞检测时输入点和输出点的一致性，从而不会发生误报漏洞的情况。

5　结束语

本文通过研究 Web 系统中 XSS 漏洞的产生原理和相关检测技术，设计了一款基于渗透测试技术的 XSS 漏洞检测系统，并实现了系统原型。该系统针对已有检测系统攻击向量复杂度低易过滤、检测流程冗余的问题，提出了一种基于 EBNF 范式的高复杂度攻击向量自动生成方法，可以有效绕过服务器端过滤。通过在攻击向量恶意脚本片段嵌入输入点信息，不仅可以简化检测流程，而且几乎避免了漏洞的误报。同时，通过进行二次爬取，进一步减少存储型漏洞的漏报，减低整体的漏洞漏报率。最后通过进行两组对比实验，证明了本系统在检测 Web 系统的 XSS 漏洞方面的可行性和有效性。

目前系统并没有涉及如何对抗 WAF 以及反爬虫措施，仅限于开发者自己的 Web 系统使用，因此存在一定的局限性。在下一步的工作中，将针对这一问题进行优化改进，扩大使用范围，完善 XSS 漏洞检测系统。

参考文献：

[1] Jeff W, Dave W. OWASP top 10-2017 [EB/OL]. (2018-03-27) [2018-04-12]. https://www. owasp. org/index. php/Top_10-2017_Top_1 0.

[2] Ryan O. WHS 2017 application security report FINAL [EB/OL]. (2017) [2018-04-12]. https://www. whitehatsec. com/resources-categor y/premium-content/web-application-stats-report-2017/.

[3] Azmi S A, Khan A R. A comprehensive research on Xss scripting attacks on different domains and their verticals [C]// Proc of the 4th International Conference on Computer Science and Network Technology. Piscataway, NJ: IEEE Press, 2015: 677-680.

[4] Barus A C, Hutasoit D I P, Siringoringo J H, *et al.* White box testing tool

prototype development [C]// Proc of the 5th International Conference on Electrical Engineering and Informatics. Piscataway, NJ: IEEE Press, 2015: 417-422.

[5] 曹黎波，曹天杰. 基于动态测试的 XSS 漏洞检测方法研究 [J]. 计算机应用与软件, 2015, 32 (8): 272-275. (Cao Libo, Cao Tianjie. Research on cross-site scripting vulnerability detection method based on dynamic testing [J]. Computer Applications and Software, 2015, 32 (8): 272-275.)

[6] Medeiros I, Neves N, Correia M. Detecting and removing Web application vulnerabilities with static analysis and data mining [J]. IEEE Trans on Reliability, 2016, 65 (1): 54-69.

[7] 余学永，江国华. 一种跨站脚本的检测方法 [J]. 小型微型计算机系统, 2015, 36 (8): 1763-1768. (Yu Xueyong, Jiang Guohua. A method for detecting cross-site script [J]. Journal of Chinese Computer Systems, 2015, 36 (8): 1763-1768.)

[8] 李威，李晓红. Web 应用存储型 XSS 漏洞检测方法及实现 [J]. 计算机应用与软件, 2016, 33 (1): 24-27, 37. (Li Wei, Li Xiaohong. Detection method for stored-XSS vulnerability in Web applications and its implementation [J]. Computer Applications and Software, 2016, 33 (1): 24-27, 37.)

[9] 顾明昌，王丹，赵文兵，等. 一种基于攻击向量自动生成的 XSS 漏洞渗透测试方法 [J]. 软件导刊, 2016, 15 (7): 173-177. (Gu Mingchang, Wang Dan, Zhao Wenbing, *et al.* A XSS vulnerability penetration test method based on attack vector automatic generation [J]. Software Guide, 2016, 15 (7): 173-177.)

[10] 吴子敬，张宪忠，管磊，等. 基于反过滤规则集和自动爬虫的 XSS 漏洞深度挖掘技术 [J]. 北京理工大学学报, 2012, 32 (4): 395-401. (Wu Zijing, Zhang Xianzhong, Guan Lei, *et al.* Technique for deep discovering XSS vulnerability based on anti-filter rules set and automatic crawler program [J]. Transaction of Beijing Institute of Technology, 2012, 32 (4): 395-401.)

[11] Rathore S, Sharma P K, Park J H, *et al.* XSS classifier: an efficient XSS attack detection approach based on machine learning classifier on SNSs [J]. Journal of Information Processing Systems, 2017, 13 (4): 1014-1028.

[12] Gupta S, Gupta B B. Cross-site scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art [J]. International Journal of System Assurance Engineering & Management, 2017, 8 (Suppl 1): 512-530.

[13] Dong G, Zhang Y, Wang X, *et al.* Detecting cross site scripting vulnerabilities introduced by HTML5 [C]// Proc of the 11th International Joint Conference on Computer Science and Software Engineering. Piscataway, NJ: IEEE Press, 2014: 319-323.